SYSTEM AND METHOD FOR DATA SYNCHRONIZATION OVER A NETWORK USING A PRESENTATION LEVEL PROTOCOL

FIELD OF THE INVENTION

[0001] The illustrative embodiment of the present invention relates generally to data synchronization, and more specifically to the synchronization of a collection of data on a device communicating with a client system with a collection of data accessible from a user session on a server system.

BACKGROUND OF THE INVENTION

[0002] Plug and Play (PnP) is a combination of hardware and software support that enables a computer system to recognize and adapt to hardware configuration changes in devices/components interfaced with the system with little or no user intervention. With Plug and Play, a user can add or remove devices dynamically, without manual configuration and without any intricate knowledge of computer hardware. For example, a user can dock a PDA or laptop and use the docking station's Ethernet card to connect to a network without changing the configuration settings for the connected device. Subsequently, the user can undock the same

PDA or laptop and use a modem to connect to the network without having to make any manual configuration changes.

[0003] Plug and play events for devices connected to a computer are typically handled by the operating system of the system with which the device is communicating. For example, a PDA tethered to a PC would have its events handled by the PC OS. The system is able to automatically load and unload device drivers to reflect the different devices attached to the system when they are docked or undocked. Also, applications are able to automatically adjust their configurations to reflect the insertion or removal of devices, e.g., PDAs. Plug and Play allows a user to change a computer's configuration with the assurance that all devices will work together and that the machine will boot correctly after the changes are made.

[0004] Unfortunately, the conventional method of handling Plug and Play events (such as “device arrival” and “device removal”) does not work particularly well where the new device is communicating with a client system connected to a server in a server-based computing system. The client system in a server-based computing system is frequently in contact with the server

system over a network via a presentation level protocol supporting a user session executing on the server. The server system deploys, manages, supports and executes applications on the servers thereby relieving the client system of the need to host and execute the applications. The server deploys a presentation level protocol and architecture such as the Independent Computing Architecture (ICA) from Citrix Systems Inc. of Fort Lauderdale, FL, the Remote Desktop Protocol (RDP) from Microsoft Corporation of Redmond, Washington and the X-Window Protocol from the X-Open Consortium. A user connected to the client system who wishes to execute an application or access resources on the server is connected in a dedicated session to the server. Conventionally however, the Plug and Play manager for the operating system of the client system handles any messages generated by devices connecting to, or removing from, the client system and the generated events do not impact the existing user session. As a result devices communicating with the client which wish to synchronize data with data available in the user session hosted by the server (e.g.: synchronize calendar entries in OUTLOOK by Microsoft Corporation) are unable to do so as the device is mapped to the client system.

BRIEF SUMMARY OF THE INVENTION

[0005] The illustrative embodiment of the present invention provides a mechanism for synchronizing data on a device communicating with a client system with data accessible from a server-hosted session. The synchronization occurs over a network connection using a presentation-level protocol to transmit communications between the client system and the server system. Devices communicating with the client system are mapped into a user session. Once the device is mapped into the server-hosted session, data on the device in communication with the client system may be synchronized with data accessible from the user-hosted session.

[0006] In an embodiment, a method for synchronizing data on a device in communication with a client system includes the step of mapping the device in communication with the client system into a user session hosted by a server. The user session includes an executing instance of an application. The server is in communication with the client system using a presentation-level protocol. The method also synchronizes a collection of data on the device in communication with the client system with a collection of

data accessible from the user session hosted by the server. The synchronization occurs as a result of the execution of the application instance.

[0007] In another embodiment, a method for synchronizing data on a device in communication with a client system which is communicating with a server using a presentation-level protocol, includes the steps of determining the identity of the device in communication with the client system and determining that the device is a member of a registered device class. The method further includes the step of creating a notification indicating that the device is in communication with the client system. The notification is directed to an instance of an application executing within a user session hosted by a server. The method also includes the step of synchronizing a collection of data on the device in communication with the client system with a collection of data accessible from the user session as a result of the execution of the application instance.

[0008] In one embodiment, a system for synchronizing data on a device in communication with a client system includes a client system executing a presentation-level protocol to communicate

with a server system The client system includes an event manager to generate event notifications based on a communication received from the device in communication with the client system. The system also includes a device in communication with the client system, the device in communication with the client including a collection of data. The system further includes a server system executing a presentation level protocol to communicate with the client system. The server system also hosts at least one user session executing an instance of an application. The application is used to synchronize the collection of data on the device in communication with the client system with a collection of data accessible from the user session.

[0009] In another embodiment, a method for synchronizing data on a device in communication with a client system includes the step of determining the identity of a device in communication with the client system via a USB connection. The method further determines that the device is a member of a registered device class and creates a notification indicating that the device is in communication with the client system. The method directs the notification to an an application executing on a server. The

execution of the application on the server synchronizes a collection of data on the device in communication with the client system with a collection of data accessible from the server.

[0010] In one embodiment, a system for synchronizing data on a device in communication with a client system includes a client system communicating with a server system. The client system includes an event manager to generate event notifications based on a communication received from the device interfaced with the client system via a USB connection. The system further includes a device in communication with the client system which includes a collection of data. The system also includes a server system communicating with the client system and executing an application used to synchronize the collection of data on the device in communication with the client system with a collection of data accessible to the server.

[0011] In a further embodiment, a method of synchronizing data on a device in communication with a client system includes the steps of providing a client system communicating with a server using a presentation-level protocol. The method further includes the step of intercepting at least one device enumeration method in

a session hosted by the server. The enumeration method enumerates the device(s) communicating with the client. The method also maps the device(s) in communication with the client system into a user session hosted by the server based on the results of the enumeration method. The user session includes an executing instance of an application. The method additionally includes the step of synchronizing a collection of data on the device in communication with the client system with a collection of data accessible from the user session as a result of the execution of the application instance.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] These and other aspects of this invention will be readily apparent from the detailed description below and the appended drawings, which are meant to illustrate and not to limit the invention, and in which:

[0013] **Figure 1A** is a block diagram of an embodiment of an environment suitable for practicing the illustrative embodiment of the present invention;

[0014] **Figure 1B** is a block diagram depicting the environment of **Figure 1A** in more detail;

[0015] **Figures 2A and 2B** are block diagrams depicting embodiments of computers useful in connection with the present invention;

[0016] **Figure 3** is a flowchart of one embodiment of the sequence of steps followed by the present invention to re-direct device generated events detected at a client system to a server system using a presentation level protocol;

[0017] **Figure 4** is a flowchart of one embodiment of the sequence of steps followed by a server in the present invention to handle the events detected in **Figure 3**;

[0018] **Figure 5** is a flowchart of one embodiment of the sequence of steps followed by the present invention to map a virtual device to a user session;

[0019] **Figure 6** is a flowchart of one embodiment of the sequence of steps followed by the present invention to send a command from an application instance executing within a session

to a virtual device communicating with a client system that has been mapped to the user session;

[0020] **Figure 7** is a flowchart of one embodiment of the sequence of steps followed by the present invention to emulate and re-direct device generated events at a client system to a server system using a presentation level protocol;

[0021] **Figure 8** is a flowchart of the sequence of steps followed by an embodiment of the present invention to synchronize a collection of data on a device in communication with a client system with a collection of data on a server within an existing user session;

[0022] **Figure 9A** is a block diagram of an embodiment of the present invention depicting the use of a proxy client in a pass-through environment;

[0023] **Figure 9B** is a block diagram of another embodiment of the present invention depicting the use of a proxy client in a pass-through environment where the proxy client is located on the server hosting the user session; and

[0024] **Figure 9C** depicts a block diagram of an embodiment of the present invention implemented without the use of a presentation-level protocol supporting architecture.

DETAILED DESCRIPTION OF THE INVENTION

[0025] The illustrative embodiment of the present invention maps devices communicating with a client system into a user session hosted by a server. Once the device communicating with the client system has been mapped into the user session, an instance of a synchronizing application executing within the user session may be utilized. The synchronizing application is used to synchronize a collection of data on the device communicating with the client system, such as a WinCE PDA, with a collection of data accessible from the user session hosted by the server.

[0026] **Figure 1A** depicts an environment suitable for practicing an embodiment of the present invention. A user 2 has access to a client system 4. The client system 4 includes a presentation-level protocol supporting architecture 6 such as ICA, RDP or X-Open Windows. The presentation-level protocol supporting architecture 6 is used to establish a connection over a network 10 with a presentation-level supporting architecture 14

included on a server 12. The presentation-level protocol supporting architecture 14 establishes user sessions, session I (16), session II(18) and session III(20). Requests from a user 2 are mapped into the appropriate session. Those skilled in the art will recognize that a single user may establish multiple sessions. Alternatively, multiple users may establish individual sessions 16, 18 and 20 on the server 12. Each session 16, 18 and 20 may include instances of one or more applications 22 and 24 available to handle client system 4 requests. A device 24 in communication with the client system 4 may generate an event 26 which is intercepted and redirected to a session 16, 18 and 20 for handling. The device 24 in communication with the client system 4 may take many forms. The device 24 may be a printer, scanner, digital camera, mobile phone, PDA, or other device exporting a serial communication interface capable of communicating with the client system 4.

[0027] Portions of the environment depicted in **Figure 1A** are shown in more detail in **Figure 1B**. An exemplary Presentation Level Protocol Supporting Architecture 6 on the client side includes an engine 30 and Protocol Stack 32. The Protocol Stack 32 includes a

Protocol Driver (PD), and Transport Driver (TD). The Presentation Level Protocol Supporting Architecture 6 also includes an OS Abstraction Layer 31, a Plug and Play Manager 34, a USB driver stack 36 and an I/O manager 38. The Presentation Level Protocol Supporting Architecture 6 further includes a control virtual driver (CTL VD) 40, used to transmit control information over a virtual control connection 44 to and from the server 12, and a Redirector virtual driver/Client COM Mapping Virtual Driver (CCM VD) 42 which is used to transmit data over a virtual communication connection 48 to and from a server 12. The CTL VD 40 registers a callback for certain events with the OS abstraction layer 31. The callback is a standing request left with the OS abstraction layer 31 by the CTL VD 40 to execute a particular instruction (such as notifying the CTL VD) in the case of the occurrence of a specified event. The OS abstraction layer 31 intercepts events generated by the PNP Manager 34 such as device arrival events generated by a printer 60 connecting to USB driver stack 36. The callback to the CTL VD 40 that is executed sends information to the server about the printer 60 and the event as set forth below in more detail.

[0028] Although **Figure 1B** depicts a printer 60 communicating with the client-side Presentation Level Protocol Supporting Architecture 6 via a USB driver stack 36, it should be understood that many other implementations are possible within the scope of the present invention. For example the device 24 communicating with the client system 4 may be any device exporting a serial interface. Accordingly, the device 24 communicating with the client system 4 may communicate using an IR serial protocol, a Bluetooth serial protocol, the IEEE 1394 protocol, FIREWIRE, the WI-FI protocol, the USB protocol, the wireless USB/ultra-wideband wireless protocol, or some other protocol as long as the client system 4 supports communications with the device 24 using the particular protocol.

[0029] The server 12 includes the server-side of the presentation level protocol supporting architecture 14 depicted in **Figure 1B**. The server-side of the presentation level protocol supporting architecture 14 supports multiple user sessions 16, 18 and 20. The depicted user session I (16) includes a server-side CTL VD 46 used to establish a virtual channel 44 with the client-side CTLVD 40 in order to transmit control information between the

client system and the server. Session I (16) also includes one or more applications 22 and 24 such as the depicted instance of the application 22 which may act as a resource for the printer 60. Separate instances of application 22 run as separate processes within each user session. Alternatively, the application 22 may be a synchronizing application, such as ACTIVESYNC from Microsoft Corporation, or HOTSYNC from PalmOne, Inc. of Milpitas, California. Those skilled in the art will recognize that Session I (16) may include many different types of applications in addition to/in place of synchronization applications without departing from the scope of the present invention.

[0030] Session I (16) also includes a Network Provider module 52 which makes an RPC call to a driver mapping service as part of the process of mapping virtual devices into a user session. The mapping process is discussed in more detail below. The Network Provider module 52 is in contact with the driver mapping service 56 which communicates with the Object Manager 58. The Driver Mapping Service 56 and the Object Manager 58 are also part of the Presentation-Level Protocol Supporting Architecture 14 but located outside the session 16. The application 22 communicates

with the I/O Manager 54. The I/O manager 54 communicates with the Object Manager 58 and the Device Redirector 50. The Device Redirector 50 is used to form a virtual communication channel 48 with the client-side CCM VD 42 for transmitting data. Also included in the Presentation Level Protocol Supporting Architecture 14 is a server-side Protocol Stack 60 including a PD and TD. The operations of the server-side components depicted in **Figure 1B** are discussed in more detail below.

[0031] Those skilled in the art will recognize that the actual components in the Presentation Level Protocol Supporting Architectures 6 and 14 will vary depending upon the architecture deployed. In the ICA architecture the protocol stacks 32 and 60 are ICA stacks. The OS Abstraction Layer 31 is a WINDOWS Abstraction Layer. Similarly, the Object Manager 58 is a WINDOWS Object Manager, the client-side Presentation Level Protocol Supporting Architecture 6 is an ICA client, and the server-side Presentation Level Protocol Supporting Architecture14 is a METAFRAME PRESENTATION SERVER in an ICA architecture. It will also be recognized that the components of an RDP architecture, X-Open architecture or other presentation level protocol supporting

architecture may vary from the components depicted herein without departing from the scope of the present invention.

[0032] In many embodiments, the client system 4 and server 12 are provided as personal computer or computer servers, of the sort manufactured by the Hewlett-Packard Corporation of Palo Alto, California or the Dell Corporation of Round Rock, TX. **Figures 2A and 2B** depict block diagrams of a typical computer 200 useful as the client system 4 and server 12. As shown in **Figures 2A and 2B**, each computer 200 includes a central processing unit 202, and a main memory unit 204. Each computer 200 may also include other optional elements, such as one or more input/output devices 230a-230n (generally referred to using reference numeral 230), and a cache memory 240 in communication with the central processing unit 202.

[0033] The central processing unit 202 is any logic circuitry that responds to and processes instructions fetched from the main memory unit 204. In many embodiments, the central processing unit is provided by a microprocessor unit, such as: the 8088, the 80286, the 80386, the 80486, the Pentium, Pentium Pro, the Pentium II, the Celeron, or the Xeon processor, all of which are

manufactured by Intel Corporation of Mountain View, California; the 68000, the 68010, the 68020, the 68030, the 68040, the PowerPC 601, the PowerPC604, the PowerPC604e, the MPC603e, the MPC603ei, the MPC603ev, the MPC603r, the MPC603p, the MPC740, the MPC745, the MPC750, the MPC755, the MPC7400, the MPC7410, the MPC7441, the MPC7445, the MPC7447, the MPC7450, the MPC7451, the MPC7455, the MPC7457 processor, all of which are manufactured by Motorola Corporation of Schaumburg, Illinois; the Crusoe TM5800, the Crusoe TM5600, the Crusoe TM5500, the Crusoe TM5400, the Efficeon TM8600, the Efficeon TM8300, or the Efficeon TM8620 processor, manufactured by Transmeta Corporation of Santa Clara, California; the RS/6000 processor, the RS64, the RS 64 II, the P2SC, the POWER3, the RS64 III, the POWER3-II, the RS 64 IV, the POWER4, the POWER4+, the POWER5, or the POWER6 processor, all of which are manufactured by International Business Machines of White Plains, New York; or the AMD Opteron, the AMD Athalon 64 FX, the AMD Athalon, or the AMD Duron processor, manufactured by Advanced Micro Devices of Sunnyvale, California.

[0034] Main memory unit 204 may be one or more memory chips capable of storing data and allowing any storage location to be directly accessed by the microprocessor 202, such as Static random access memory (SRAM), Burst SRAM or SynchBurst SRAM (BSRAM), Dynamic random access memory (DRAM), Fast Page Mode DRAM (FPM DRAM), Enhanced DRAM (EDRAM), Extended Data Output RAM (EDO RAM), Extended Data Output DRAM (EDO DRAM), Burst Extended Data Output DRAM (BEDO DRAM), Enhanced DRAM (EDRAM), synchronous DRAM (SDRAM), JEDEC SRAM, PC100 SDRAM, Double Data Rate SDRAM (DDR SDRAM), Enhanced SDRAM (ESDRAM), SyncLink DRAM (SLDRAM), Direct Rambus DRAM (DRDRAM), or Ferroelectric RAM (FRAM).

[0035] In the embodiment shown in **Figure 2A**, the processor 202 communicates with main memory 204 via a system bus 220 (described in more detail below). **Figure 2B** depicts an embodiment of a computer system 200 in which the processor communicates directly with main memory 204 via a memory port. For example, in **Figure 2B** the main memory 204 may be DRDRAM.

[0036] **Figures 2A and 2B** depict embodiments in which the main processor 202 communicates directly with cache memory 240

via a secondary bus, sometimes referred to as a "backside" bus. In other embodiments, the main processor 202 communicates with cache memory 240 using the system bus 220. Cache memory 240 typically has a faster response time than main memory 204 and is typically provided by SRAM, BSRAM, or EDRAM.

[0037] In the embodiment shown in **Figure 2A**, the processor 202 communicates with various I/O devices 230 via a local system bus 220. Various buses may be used to connect the central processing unit 202 to the I/O devices 230, including a VESA VL bus, an ISA bus, an EISA bus, a MicroChannel Architecture (MCA) bus, a PCI bus, a PCI-X bus, a PCI-Express bus, or a NuBus. For embodiments in which the I/O device is a video display, the processor 202 may use an Advanced Graphics Port (AGP) to communicate with the display. **Figure 2B** depicts an embodiment of a computer system 200 in which the main processor 202 communicates directly with I/O device 230b via HyperTransport, Rapid I/O, or InfiniBand. **Figure 2B** also depicts an embodiment in which local busses and direct communication are mixed: the processor 202 communicates with I/O device 230a using a local

interconnect bus while communicating with I/O device 230b directly.

[0038] A wide variety of I/O devices 230 may be present in the computer system 200. Input devices include keyboards, mice, trackpads, trackballs, microphones, and drawing tablets. Output devices include video displays, speakers, inkjet printers, laser printers, and dye-sublimation printers. An I/O device may also provide mass storage for the computer system 200 such as a hard disk drive, a floppy disk drive for receiving floppy disks such as 3.5-inch, 5.25-inch disks or ZIP disks, a CD-ROM drive, a CD-R/RW drive, a DVD-ROM drive, tape drives of various formats, and USB storage devices such as the USB Flash Drive line of devices manufactured by Twintech Industry, Inc. of Los Alamitos, California.

[0039] In further embodiments, an I/O device 230 may be a bridge between the system bus 220 and an external communication bus, such as a USB bus, an Apple Desktop Bus, an RS-232 serial connection, a SCSI bus, a FireWire bus, a FireWire 800 bus, an Ethernet bus, an AppleTalk bus, a Gigabit Ethernet bus, an Asynchronous Transfer Mode bus, a HIPPI bus, a Super HIPPI bus, a

SerialPlus bus, a SCI/LAMP bus, a FibreChannel bus, or a Serial Attached small computer system interface bus.

[0040] General-purpose desktop computers of the sort depicted in **Figures 2A and 2B** typically operate under the control of operating systems, which control scheduling of tasks and access to system resources. Typical operating systems include: MICROSOFT WINDOWS, manufactured by Microsoft Corp. of Redmond, Washington; MacOS, manufactured by Apple Computer of Cupertino, California; OS/2, manufactured by International Business Machines of Armonk, New York; and Linux, a freely-available operating system distributed by Caldera Corp. of Salt Lake City, Utah, among others.

[0041] For embodiments in which the device in communication with the client system 24 is a mobile device, the client device may be a JAVA-enabled cellular telephone, such as the i50sx, i55sr, i58sr, i85s, i88s, i90c, i95cl, or the im11000, all of which are manufactured by Motorola Corp. of Schaumburg, Illinois, the 6035 or the 7135, manufactured by Kyocera of Kyoto, Japan, or the i300 or i330, manufactured by Samsung Electronics Co., Ltd., of Seoul, Korea. In other embodiments in which the device 24 in

communication with the client system is mobile, it may be a personal digital assistant (PDA) operating under control of the PalmOS operating system, such as the Tungsten W, the VII, the VIIx, the i705, all of which are manufactured by palmOne, Inc. of Milpitas, California. In further embodiments, the device 24 in communication with the client system 4 may be a personal digital assistant (PDA) operating under control of the PocketPC operating system, such as the iPAQ 4155, iPAQ 5555, iPAQ 1945, iPAQ 2215, and iPAQ 4255, all of which manufactured by Hewlett-Packard Corporation of Palo Alto, California, the ViewSonic V36, manufactured by ViewSonic of Walnut, California, or the Toshiba PocketPC e405, manufactured by Toshiba America, Inc. of New York, New York. In still other embodiments the device in communication with the client system 24 is a combination PDA/telephone device such as the Treo 180, Treo 270 or Treo 600, all of which are manufactured by palmOne, Inc. of Milpitas, California. In still a further embodiment, the device in communication with the client system 24 is a cellular telephone that operates under control of the PocketPC operating system, such as the MPx200, manufactured by Motorola Corp.

[0042] As previously mentioned, the illustrative embodiment of the present invention re-directs to a server 12 plug and play events and other types of events such as emulated plug and play events generated about the device in communication with the client system 24. The events are redirected over a network 10 using a presentation level protocol. **Figure 3** is a flowchart of the client-side sequence of steps followed by an embodiment of the present invention to redirect to an existing user session on a server a detected event caused by a device 24 in communication with the client system 4. Although, the description contained herein discusses the re-direction of detected plug and play events (and emulated fake plug and play events) it should be understood that other types of events may also redirected without departing from the scope of the present invention.

[0043] The client-side sequence of steps in **Figure 3** begins with the Control VD 40 registering a callback to be notified of certain events with the OS abstraction layer 31 (step 300). The callback may specify that the Control VD 40 is only interested in certain device classes that generate an event. Alternatively, the callback registration may request that the Control VD 40 be notified

only in the case of a particular type of event (e.g.: device arrival and removal). Similarly, the Control VD 40 may request that it be notified only of events that are generated by devices connected to a particular port (e.g.: only notify for devices connected to a USB port). Those skilled in the art will recognize that additional or other parameters may be specified in the callback registration. When a device in communication with the client system is docked at a USB port (step 302), the PNP manager 34 issues a device arrival event (step 304) which is intercepted by the OS abstraction layer 31. The event notification includes information such as a GUID for the device (a globally unique ID), a vendor ID, a product ID, the type of event and a device name. Upon intercepting the event notification, the OS abstraction layer 31 determines if a registration exists for the detected event, and, if so, executes a callback to the Control VD 40 informing the Control VD of the occurrence of the event for which it registered. The callback passes the event information to the Control VD (step 306).

[0044] Upon receiving the callback, the Control VD 40 queries a table to identify a redirector VD (CCM VD 42) and sends a Bind request to the redirector VD (CCM VD) (step 308). The CCM VD

42 provides a logical COM port number for the Virtual Communication Channel (referred to as a COM context or virtual context herein) and binds it to the event information indicated in the request (GUID, Vendor ID, Product ID, Device Name, etc.). The bound request is returned to the Control VD 40 (step 310). The binding (the event information plus the COM context for the Virtual Communication Channel 48) is then sent to the server 12 (step 312). The Control VD 40 receives response from the server 12 indicating the success or failure of the mapping of the virtual device in the respective session (step 314). Upon failure, the Control VD 40 notifies the CCM VD 42 to unbind the virtual context. In the event of a successful mapping operation, subsequently, the CCM VD 42 receives a command from an Application 22 running in a user session 16, 18 and 20 on the server 12 that is addressed to the Redirector VD/CCM VD 42 at the virtual context. For example, the command may be a command to open the referenced virtual COM port assigned in the binding process (the virtual COM number represented by the COM context) to the device in communication with the client system . Once the virtual COM port has been opened, subsequent I/O requests for the device in communication

with the client system are received on the virtual COM port referenced in the binding (step 316).

[0045] **Figure 4** is a flowchart of one embodiment of the sequence of steps followed by a server in the present invention to handle the events detected in **Figure 3**. The sequence begins when the Control VD 46 on the server 12 receives the bound PNP event notification (step 320). The Control VD then requests the device in communication with the client system be mapped into the session. (step 322). The details of the mapping request are discussed further in reference to **Figure 5** below. If the mapping request fails (step 321), a failure message is transmitted back to the CCM VD 42 so that the device can be unbound (step 322). If the mapping request succeeds (step 321), the CTL VD 46 broadcasts the PNP event within the current session (step 324). In another embodiment, the PnP device registration API is intercepted on the server 12, and only apps that have registered for PnP events are notified. An application 22 within the session receives the PNP broadcast event (step 326) and sends an Open command to the server OS referencing the virtual device bound into the event notification (step 328). The server OS, with the help of the I/O Manager 54 and Object

Manager 58, and based on the syntax of the symbolic link established in the mapping (step 322), redirects it to the Citrix Device Redirector 50 step 328), which in turn redirects it to the corresponding session and I/O Redirector VD, again based on the syntax of the symbolic link (step 330). The steps taken by the application 22 to send a command to the device in communication with the client system are discussed further in **Figure 6** below. The command may be an "Open" command and be directed to the particular COM communication port. The command is forwarded to the I/O Redirector VD (CCM VD) 42 on the client side for further processing where execution of the command will be attempted.

[0046] The mapping process engaged in by the server-side presentation-level protocol supporting architecture 14 is now discussed in further detail. **Figure 5** depicts the sequence of steps followed by an embodiment of the present invention to map the received event notification to a virtual device in an existing user session. The sequence begins following the server side Control VD 46 receiving the bound PNP event notification and requesting mapping of the virtual device into the user session (step 320 of **Figure 4**). The CTL VD 46 generates unique names for the server

and remote client using the information in the event notification (step 338). The unique name for the server may be based on the information received in the event notification, the GUID, virtual context, or context ID. For example, if the GUID is WinCE interface class, the name generated may be WCEUSBSHxxx in order where xxx is the COM context. The unique name for the remote client may be based on the virtual context (VD name and context). The CTL VD 46 loads the driver mapping network provider module 52 and provides the unique names (step 340). The network provider module 52 places an RPC call to the driver mapping service module 56 (step 342). The driver mapping service 56 generates a path for the virtual device and requests the Object Manager 58 create a symbolic link to the virtual device in the user session 16 using the name information (step 344). The symbolic link is entered into an internal table in the session (step 346). The I/O Manager 54 will later use the first part of the symbolic link information to perform redirection. The server-side Control VD 46 then broadcasts a PNP notification within the session which includes the symbolic link to the virtual device (step 324 of Figure 4). Those skilled in the art will recognize that although a broadcast by the Control VD of the event notification has been discussed above, a callback registration

process by which applications register within the session to be notified of the occurrence of a specified event may also be employed without departing from the scope of the present invention. In such a case, only those applications within the session that previously requested notification of the occurrence of the specified event would be notified following the mapping of the virtual device within the session.

[0047] After the device in communication with the client system has been mapped into the user session 16 and the event notification broadcast within the session, applications running within the session are able to respond to the event. Upon receiving the PNP event notification, the application 22 attempts to open a local device name of the device so that commands and data may be sent to the device. If the device in communication with the client system is a WinCE device, ACTIVESYNC expects a specific name in the form of WCEUSBSHxxx. Only one device is supported by ACTIVESYNC so the unique names in the form WCEUSBSHxxx are generated in order in the present invention. For other synchronization applications, unique names that are a combination

of device interface class GUID and V-Context where both the virtual device name and the device interface class are unique may be used.

[0048] **Figure 6** is a flowchart depicting the sequence of steps followed by an embodiment of the present invention to respond to an event notification. Upon receiving the PNP event notification, the application 22 attempts to open a local device name of the device so that commands and data may be sent to the device. The sequence begins with the Application 22 sending an Open command to a local I/O manager 54 (step 360). The I/O manager 54 sends a name resolution request to the Object Manager 58 (step 362). The Object Manager 58 returns a redirection path to the I/O manager 54 (step 364) which forwards the Open command and the redirection path to the Device Redirector 50 (step 366). The Device Redirector 50 resolves the path to a particular session and an I/O Redirection VD 50 (CCM VD) (step 368). The Device Redirector 50 then transmits a command to the CCM VD 42 for execution (step 370). For example, the application may transmit an "open" request for virtual COM port 3, which will be transmitted to the client side I/O Redirector VD (CCM VD 42) for execution. The client CCM VD 42 will apply the Open command to the actual device

bound to the virtual COM port. Once the virtual COM port is open, it will be used to transmit data between the server side application 22 and the device in communication with the client system 24 over the virtual communication channel 48. It should be understood that other commands other than "open" may be sent without departing from the scope of the present invention.

[0049] Device removal works similarly to device addition. The detected removal event is sent with a release device request to the Redirector VD 42. The Redirector VD 42 binds the COM context to the notification. The bound event is sent to the server where it is parsed. At the server, the virtual device instance is unmapped from the user session 16. The MPS sends a release command back to the CTL VD on the client system. The CTL VD forwards the release command to the CCM VD which releases the assigned COM context, and records the COM port availability.

[0050] In addition to detecting PNP events from a device 24 communicating with the client system 4, the illustrative embodiment of the present invention also enables devices previously connected to be mapped into a user session by emulating a plug and play event and then handling the emulated

event as outlined above. **Figure 7** depicts the sequence of steps followed by an embodiment of the present invention to emulate a plug and play event for a previously connected device in order to map the device into the existing user session. The sequence begins with the determination of devices in communication with the client system (step 380). This may be determined by the client system operating system enumerating the communicating devices. For each device, the client system then emulates an event(step 382). The context ID is bound to each event (step 384) as outlined above and the emulated event information, including the bound context ID is sent to the server over the control communication channel (step 386). Subsequently following the mapping of the device into the user session, the client system may receive a command from an application instance running in the user session directed to the virtual driver for the device (step 388).

[0051] The illustrative embodiment of the present invention also enables the synchronizing of a collection of data on the device in communication with the client system with a collection of data on the server. For example, the data in a user's OUTLOOK (from Microsoft Corporation) application on a PDA may be synchronized

with a user's OUTLOOK data on a server using a synchronizing application executing within a user session rather than synchronizing the OUTLOOK application with data from the client system to which the PDA is communicating.

[0052] **Figure 8** depicts the sequence of steps followed by an embodiment of the present invention to synchronize a collection of data on a device in communication with a client system with a collection of data on a server within an existing user session. The sequence begins with the notification of the presence of a device in communication with a client system being bound to a virtual context at the client system as discussed above (step 400). The binding is then sent to the server (step 402) where the device is mapped to a user session (step 404). An instance of a synchronizing application which is executing in the user session is then selected by a user. The synchronizing application may be ACTIVESYNC from Microsoft Corporation, HOTSYNC from PalmOne or another type of synchronizing application. The synchronizing application then compares a specified collection of data on the device 24 in communication with the client system 4 with a collection of data accessible from the user session (step 408). For

example, ACTIVESYNC may be used to synchronize a calendar in OUTLOOK on the device24 in communication with the client system 4 with the calendar in an instance of OUTLOOK on a user desktop within the session. Once the collections of data have been compared, they are synchronized by the application (step 410).

[0053] ActiveSync is the standard Microsoft application that provides synchronization and device management for a WinCE-based PDA device (PocketPC).

[0054] ActiveSync is both session (multi-user) unaware and multi-host (multi-computer) unaware. ActiveSync processes use windows sockets for inter-process communication (IPC) between themselves, and other processes, e.g., *explorer.exe*, *outlook.exe*. Sockets are used even for communication between different threads of an ActiveSync process. A combination of local host loop-back address (127.0.0.1), and all interfaces (0.0.0.0, i.e., INADDR_ANY) are used. In addition, ActiveSync always runs on the console, as even if removed from the startup sequence, ActiveSync recreates itself in the startup the next time it is launched. Without hooking, when ActiveSync runs in multiple user sessions, socket connections collide. Even if a single ActiveSync instance is run in a remote

session, collision with the local console session's ActiveSync instance occurs. Accordingly, the illustrative embodiment of the present invention hooks socket calls to avoid collisions.

[0055] Windows socket communications of ActiveSync processes and *explorer.exe* (in communication with ActiveSync processes only), are hooked by the present invention to avoid collisions. In session, special virtual IP address hooking and virtual loop-back address hooking are performed. The *Virtual loop-back address* hooking adds the session ID to the local-host loop-back address (127.0.0.1) represented as a ULONG. Virtual loop-back addresses are and have to be unique with only one address per server. Virtual loop-back addresses are also used as Virtual IP addresses. Virtual IP address hooking is performed only for non-zero ports, i.e., for local communications of ActiveSync. Socket communications are also hooked on the console.

[0056] Examples of the hooking performed by the illustrative embodiment of the present invention are set forth below.

[0057] If configured for a process, *virtual loop-back address* and/or *virtual IP address hooking*/modification may done in all of the following intercepted methods:

[0058] Bind

[0059] **Virtual loop-back**: If the local address is the local-host loop-back address (127.0.0.1), it is replaced with a virtual loop-back address.

[0060] **Virtual IP**: If the local address is INADDR_ANY (0.0.0.0 all interfaces), it is replaced with a virtual IP address.

[0061] In most implementations, the (local) port is not checked but a configuration may be added to make the replacement contingent (or not) upon the port being non-zero.

[0062] Connect

[0063] **Virtual loop-back**: If the remote address is the local-host loop-back address, it is replaced with a virtual loop-back address.

[0064] **Virtual IP**: If no local address is specified in the *Connect* call (INADDR_ANY), then a force-*Bind* is done to the virtual IP address as the local address.

[0065] Although the (local) port is not usually checked, a configuration can be added to make the replacement contingent (or

not) upon the port being non-zero (*Bind* is not affected, since *local* port is zero). *Bind* is not forced (no explicit *Bind*) in case of loop-back address hooking, since, otherwise, the Winsock API returns an error "Host Unreachable".

[0066] SendTo

[0067] Same as *Connect*.

[0068] Accept

[0069] **Virtual loop-back**: If the remote address is the virtual loop-back address for the process, it may be replaced with the local-host loop-back address.

[0070] **Virtual IP**: No action.

[0071] Although the port is not usually checked, a configuration can be added to make the replacement contingent (or not) upon the port being non-zero.

[0072] *GetPeerName* (for the remote socket)

[0073] Same as *Accept*.

[0074] *GetSockName* (for the local socket)

[0075] **Virtual loop-back**: Same as *Accept* and *GetPeerName*, however with respect to the local address.

[0076] **Virtual IP**: No action, i.e., same as *Accept* and *GetPeerName*.

[0077] The *explorer.exe* is hooked only in a session so that it affects only local-host loop-back socket communications of explorer.exe with ActiveSync. It does not affect communications with other processes as well as remote socket communications of *explorer.exe*.

[0078] On the console special virtual IP address hooking is performed so that INADDR_ANY is substituted with the *local host loop-back address* (127.0.0.1, which is the same as *virtual loop-back address* for SID = 0) if the port is not 0 (zero).

[0079] Those skilled in the art will recognize that other hooking operations may be performed in addition to, or in place of, those outlined above without departing from the scope of the present invention. Alternatively, an unused port may be substituted for a targeted port as an alternative method of addressing the

ACTIVESYNC loop-back issue without departing from the scope of the present invention.

[0080] In addition to the previously depicted client-server architecture, the illustrative embodiments of the present invention allow for a "pass-through" architecture where the device in communication with a client system that is connected over a network to session running on a server that is hosting a proxy client.. **Figure 9A** depicts a WinCE enabled PDA 500 connected to client system 501, which accesses a proxy client 520 executing in a session 505 hosted on a server 510. The client system 501 may be an ICA client, which establishes a Control VD 503 and an I/O Redirector VD 504 with the Server 510. The proxy client 520 is connected over a network to another session 560 running on a second server 550. The proxy client 520 may be an ICA client which establishes a control virtual channel 530 and an I/O Redirector VD 540 with the second server 550. An instance of application 570 executes within session 560. The WinCE PDA 500 may be mapped into the second server 550 consistent with the mechanisms outlined above.

[0081] Similarly, **Figure 9B** shows an alternative architecture where both the proxy client and user session running an instance of the application are supported by the same server. In **Figure 9B**, the device 600 is in communication with client system 601, which is connected over a network to session 605 running on server 610. Session 605 runs a proxy client 620 connected to a user session 630 running on the same server 610. User session 630 includes an instance of executing application 640. Device 600 may be mapped into the user session 630 on server 610 via the proxy client 620.

[0082] In another embodiment, **Figure 9C** depicts a block diagram of an embodiment of the present invention implemented without the use of a presentation-level protocol supporting architecture. In **Figure 9C**, the device 700 is in communication with client system 710 which is connected over a network to server 720. Server 720 hosts an executing application 740. Events occurring at the device 700 are re-directed to the server 720 and the application 740. Those skilled in the art will recognize that additional implementations and alternative architectures are also possible within the scope of the present invention.

[0083] In another embodiment, the invention includes interception of the device enumeration methods in a server session (which may be performed via a hook DLL), and their redirection (which may be performed via an RPC call) to the Control VD 46 running in the server session (See **Figure 1B**). The redirected methods will enumerate only devices mapped in the respective session based on the information in the internally maintained table of virtual device mappings, rather than devices attached to the server console itself. Thus an application running in a server session is able to enumerate devices that have already been mapped prior to launching of the application, i.e., devices the PnP event notifications of which the application has missed.

[0084] It will be appreciated by those skilled in the art that although the synchronization process has been discussed and illustrated herein with reference to virtualizing serial channels and COM ports, other types of communications may be virtualized in the manner discussed herein as part of the synchronization process. For example, USB communications may be virtualized to perform the synchronization communications in an alternate embodiment.

[0085] The present invention may be provided as one or more computer-readable programs embodied on or in one or more articles of manufacture. The article of manufacture may be a floppy disk, a hard disk, a compact disc, a digital versatile disc, a flash memory card, a PROM, a RAM, a ROM, or a magnetic tape. In general, the computer-readable programs may be implemented in any programming language. Some examples of languages that can be used include C, C++, C#, or JAVA. The software programs may be stored on or in one or more articles of manufacture as object code.

[0086] While the invention has been shown and described with reference to specific preferred embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the following claims.